

Tópicos de Desenvolvimento de Software

Desenvolvimento de Guia Turístico

10 de maio de 2024

Daniel Du
University of Minho
pg53751

José Pedro Fonte
University of Minho
a91775

Ricardo Lucena
University of Minho
pg54187

SUMÁRIO

Desenvolvimento de uma aplicação em Android nativo que funciona como guia turístico na zona de Braga usando o Android Studio . Iremos abordar as funcionalidades desenvolvidas, resultados obtidos, bem como todas as decisões tomadas ao longo do projeto.

Introdução

Neste trabalho prático foi-nos apresentado o desafio de construir uma aplicação que tem como intuito disponibilizar trilhos, conjunto de pontos de interesse/turísticos, para um utilizador percorrer. A aplicação teria de ser desenvolvida em *android studio*, uma tecnologia que fomos aprofundando ao longo das aulas práticas dadas durante o semestre.

A plataforma, denominada de *Braguia*, apresenta várias funcionalidades, desde oferecer uma lista de trilhos para um utilizador percorrer, até à integração dos mesmos no *google maps*. Todo o desenvolvimento da aplicação foi facilitado pela equipa docente que forneceu uma *API* de onde teríamos de retirar todas as informações relevantes para a construção da aplicação, quer sejam trilhos, pontos de interesse, ou credenciais para os utilizadores. Para além da *API* a equipa docente forneceu uma lista de requisitos obrigatórios, funcionais, e não funcionais, que nos guiaram na realização do trabalho.

Ao longo deste relatório abordaremos todas as estratégias e decisões que a equipa teve, a estrutura do trabalho, e as dificuldades ao longo do projeto.

Detalhes de implementação

Estrutura do projeto

A imagem presente nos anexos ([A - Estrutura do Projeto](#)) apresentamos a estrutura do projeto.

Model

Dentro do **model** temos três packages - **App, Users e Trails**.

Como subjacente aos nomes, dentro de cada um desses packages estão as classes responsáveis pela lógica de negócios da aplicação.

Dentro do package **App**:

1. Classe **App.java** : Dentro desta classe representamos os dados e tratamos a lógica de negócios de tudo que seja sobre a app em si, isto é, o nome da aplicação, a descrição, contactos, parceiros, redes sociais, entre outros.
2. Interface **AppAPI.java** : Interface que executa os pedidos HTTP GET à API.
3. Interface **AppDAO** : Interface que contem operações para acesso aos dados na base de dados.

4. Classe **Partners.java** : Classe responsável por representar os dados e tratar da lógica de negócios de tudo que seja sobre os parceiros da aplicação. Por exemplo, o nome, a descrição, o contacto e mail do parceiro, entre outros.
5. Classe **Socials** : Classe responsável por representar os dados e tratar da lógica de negócios de tudo que seja sobre as redes sociais da aplicação. Por exemplo, o nome e o URL Link da rede social, entre outros.
6. Classe **Contacts** : Classe responsável por representar os dados e tratar da lógica de negócios de tudo que seja sobre os contactos da aplicação. Por exemplo, o nome, a descrição, o contacto e mail do contacto, entre outros.
7. Classe **ConvertAppTable.java** : Classe que contem métodos usados para converter datatypes complexos, como por exemplo listas de objetos do tipo Contacto ou Partners, para e de um formato que as librarias Room e Android SQLite suportem.

Dentro do **package Trails**:

1. Classe **Conteudo.java** : Dentro desta classe representamos os dados e tratamos a lógica de negócios de tudo que seja sobre a app em si, isto é, o nome da aplicação, a descrição, contactos, parceiros, redes sociais, entre outros.
2. Interface **ConvertEdgeTable.java / ConvertPontoTable / ConvertTrailTable** : Classes que contem métodos usados para converter DataTypes complexos para e de um formato que as librarias Room e Android SQLite suportem.
No caso da *ConvertEdgeTable* transforma um objeto do tipo Ponto para uma String e vice versa.
No caso da *ConvertPontoTable* transforma listas de RelPin e Conteudo para string e vice-versa.
3. Classe **Edge.java** : Classe responsável por representar o caminho de um ponto de partida a um ponto de chegada.
4. Classe **Ponto** : Classe responsável por representar os dados e tratar da lógica de negócios de tudo que seja sobre os Pontos de interesse. Por exemplo, o nome, a descrição, a latitude e longitude, entre outros.
5. Classe **RelPin.java** : Classe responsável por representar os *rel_pin* dentro de um determinado Ponto. Tudo o que é guardado nesta classe, nunca é usado ao longo do trabalho.
6. Classe **Trail.java** : Classe responsável por representar os dados e tratar da lógica de negócios de tudo que seja sobre um trilho. Por exemplo, o nome, a descrição, duração, dificuldade, mídia, e lista de *Edges* entre outros.
7. Interface **TrailAPI.java** : Interface que executa os pedidos HTTP GET à API.
8. Classe **TrailDAO.java** : Interface que contem operações para acesso aos dados nas base de dados.

Dentro do **package Users**:

1. Classe **User.java** : Classe responsável por representar os dados e tratar da lógica de negócios de tudo que seja sobre um user. Por exemplo, o primeiro e último nome, o email, o tipo, entre outros.
2. Inferface **UserAPI.java** : Interface que executa os pedidos HTTP GET e POST à API.
3. Interface **UserDAO.java** : Interface que contem operações para acesso aos dados nas base de dados.

Repositories

1. **AppRepository.java** : Repositório responsável por gerir a comunicação entre a aplicação e a fonte de dados - API, Cache e Disco - sobre os dados relacionados apenas sobre a “app”, por exemplo, pedir dados sobre um parceiro, contacto e redes sociais, entre outros.
2. **TrailRepository.java** : Repositório responsável por gerir a comunicação entre a aplicação e a fonte de dados - API, Cache e Disco - sobre os dados relacionados apenas sobre um “Trilho”, por exemplo, pedir a mídia de um trilho, pontos de interesse de um trilho, entre outros.
3. **UserRepository.java** : Repositório responsável por gerir a comunicação entre a aplicação e a fonte de dados - API, Cache e Disco - sobre os dados relacionados apenas sobre um “user”, por exemplo, um pedido de login e logout

Ui

Dentro do **package activities** : Uma vez que apenas temos uma activity neste package só temos a *MainActivity*.

Dentro do **package fragments** : Dentro deste packages temos as várias classes responsáveis por apresentar os dados ao utilizador (à interface).

ViewModel

Classe **AppViewModel.java** : Camada intermediária entre a ui e o repository da “app”.

Classe **TrailsViewModel.java** : Camada intermediária entre a ui e o repository dos “trilhos”.

Classe **UserViewModel.java** : Camada intermediária entre a ui e o repository dos “users”.

Utils

Este package foi criado com o propósito de auxiliar algumas funcionalidades da aplicação, como por exemplo o serviço de localização.

As classes **Geofencing.java** e **GeofencingBroadcastReceiver.java** são classes responsáveis definir uma área envolvente de um ponto de interesse, e caso um utilizador entre nessa área a aplicação emite uma notificação a informar que está próximo de um ponto de interesse.

As classes **LocationForegroundService.java** e **NotificationHelper.java** são as classes responsáveis pela obtenção da localização atual do utilizador e pela implementação das notificações, respetivamente.

Soluções de implementação

Bibliotecas/dependências utilizadas

- Retrofit: Biblioteca para realizar requisições de rede de forma simples e eficiente, facilitando a comunicação com APIs RESTful.
- Picasso: Biblioteca para o carregamento e exibição de imagens de forma eficiente, lidando automaticamente com o cache e a manipulação de imagens.
- Room: Biblioteca oficial do Android para acesso a banco de dados SQLite, que oferece um ORM conveniente para acesso e manipulação de dados.
- Google Play Services Location: Dependência para aceder aos serviços de localização fornecidos pelo Google Play Services, permitindo que a aplicação obtenha a localização do dispositivo.
- ExoPlayer: Utilizada para reprodução de áudio e vídeo, oferecendo suporte a diversos formatos de media e funcionalidades avançadas, como streaming adaptativo.

Padrões de software utilizados

MVVM

Para o desenvolvimento do projeto seguimos o padrão MVVM - *Model View ViewModel*. Este padrão define uma separação clara entre a camada de lógica de negócios e a camada de apresentação, permitindo a construção de aplicações robustas e escaláveis, mas que também pode levar à perda de performance devido à ligação bi-direcional entre o *view* e o *viewmodel*. Para além disso, este modelo não é o melhor quando pretendemos construir aplicações mais pequenas, o que não se aplica ao nosso caso. Vamos começar por definir as partes envolventes desta arquitetura:

- **Model:** Camada não visual e sem conhecimento sobre a interface do utilizador responsável por representar os dados e a lógica de negócios da aplicação.
- **View:** Responsável por exibir os dados do *Model* de forma fácil e clara para os utilizadores, desta forma permite que hajam interações entre utilizador-máquina, que posteriormente são encaminhadas para o *ViewModel*.
- **ViewModel:** Atua como intermediário entre o *Model* e a *View*. Implementa propriedades e comandos para qual a *View* consegue se ligar ao *Model*. Para além disso, contém a lógica de apresentação e a lógica de interação do utilizador.

No trabalho prático fomos criando outras pastas no trabalho para facilitar o desenvolvimento, tais como:

- **Repositories:** Camada responsável por reunir e disponibilizar o acesso aos dados de diferentes fontes, isto é, de dados provenientes de várias API's, de dados em Cache e até mesmo armazenados localmente.
- **Utils:** Nesta diretoria temos classes auxiliares para a criação de notificações, quer seja a classe

Singleton

Este padrão foi implementado na classe ***GuideDatabase.java***.

Este padrão assegura que uma classe tenha apenas uma instância e fornece um ponto de acesso global para essa instância.

```
@Database(entities = {Trail.class, User.class, App.class, Contacts.class,
Partners.class, Socials.class, Edge.class, Ponto.class, Conteudo.class, RelPin.class},
version = 970)
public abstract class GuideDatabase extends RoomDatabase {
    private static final String DATABASE_NAME = "BraGuide";

    public abstract TrailDAO trailDAO();
    public abstract UserDao userDao();
    public abstract AppDAO appDAO();

    public static volatile GuideDatabase INSTANCE = null;

    public static GuideDatabase getInstance(Context context) {
        if (INSTANCE == null) {
            synchronized (GuideDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context, GuideDatabase.class,
DATABASE_NAME)
                        .fallbackToDestructiveMigration()
                        .build();
                }
            }
        }
    }
}
```

```

        return INSTANCE;
    }
}

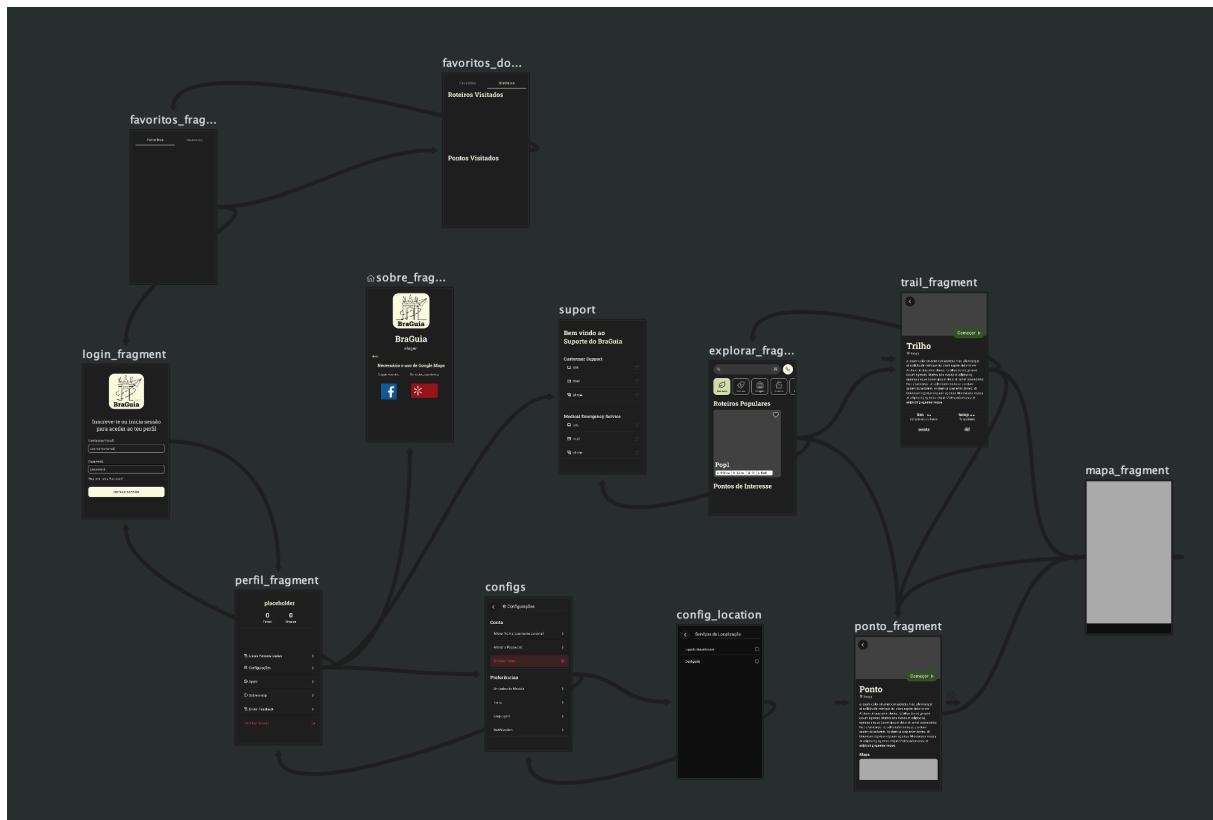
```

O código acima escrito representa o método para a criação da única instância.

Mapa de navegação de GUI

Dada a experiência de utilizador e interface desenhada pela equipa, a solução idealizada é composta por apenas uma *activity* com diferentes *fragments*. Esta decisão tem por base o princípio de o utilizador conseguir aceder a certas funcionalidades sem necessitar de estar autenticado.

Desse modo o mapa de navegação da GUI tem o seguinte aspeto:



A navegação da aplicação dá-se por uma *bottom navigation* com *fragments* associados aos *tabs*, desse modo existem cinco *tabs*:

- Sobre : fragment_sobre
- Explorar : fragment_explorar
- Mapa : fragment_mapa
- Favoritos : fragment_favoritos e fragment_historico
- Perfil : fragment_perfil(perfil de utilizador) e fragment_login (autenticação)

Os restantes *fragments* são páginas que são acedidas através de outros *fragments*, como por exemplo:

- fragment_trail: acedido através do tab explorar
- fragment_ponto_interesse : acedido através do tab explorar, do mapa e de uma página de trail.
- fragment_configs e fragment_config_location : acedidos através do perfil.
- fragment_support: acedido através do tab explorar e do perfil.

Funcionalidades

O grupo conseguiu implementar as seguintes funcionalidades, de acordo com o requisitos do enunciado.

- A aplicação deve possuir uma página inicial onde apresenta as principais funcionalidades do guia turístico, descrição, etc.



Figura 1: Tab Sobre

- A aplicação deve mostrar num ecrã, de forma responsiva, uma lista de roteiros disponíveis;

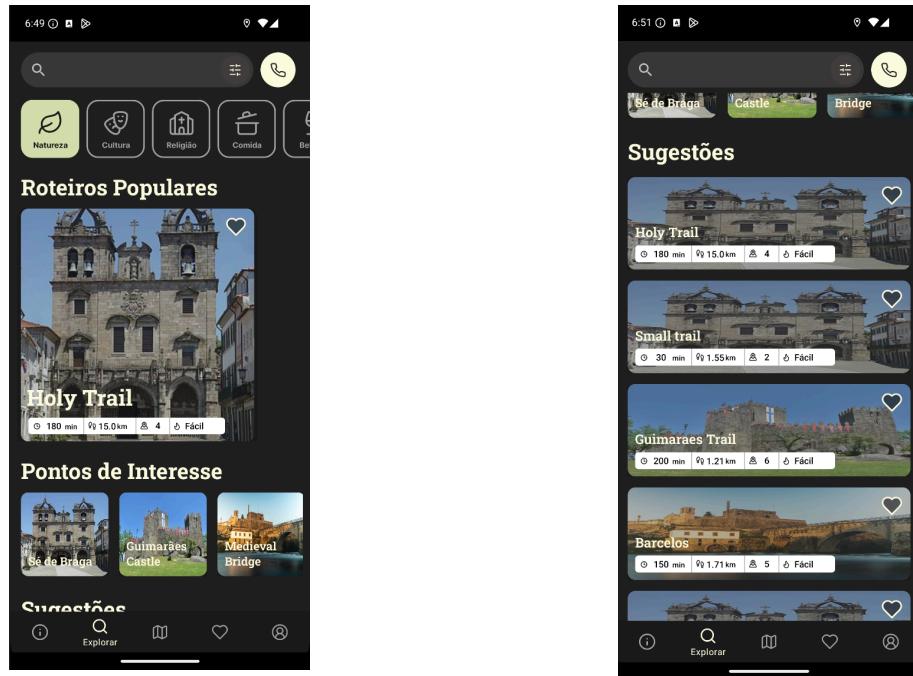


Figura 2: Tab Explorar

- A aplicação deve permitir efetuar autenticação;

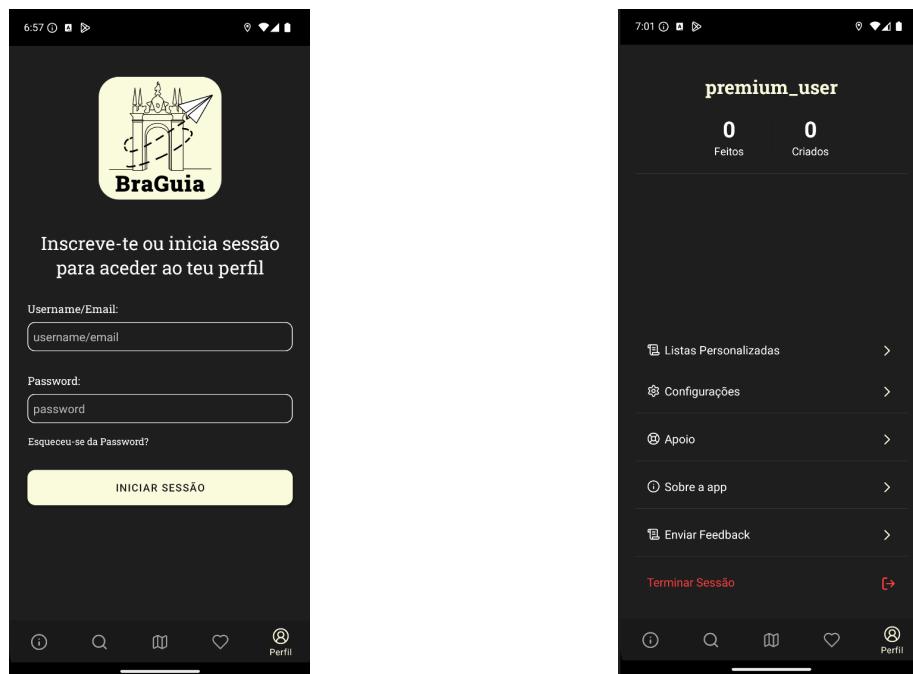


Figura 3: Tab Perfil

- A aplicação deve suportar 2 tipos de utilizadores: utilizadores standard e utilizadores premium;

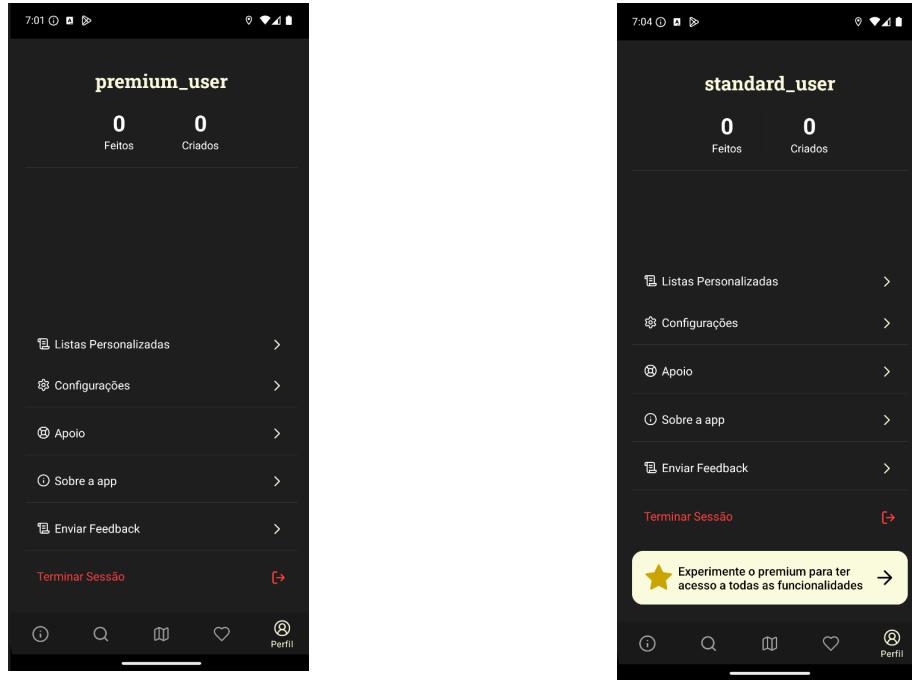


Figura 4: Tipos de Utilizador

- A aplicação deve assumir que o utilizador tem o Google Maps instalado no seu dispositivo (e notificar o utilizador que este software é necessário); **A app informa que o Google Maps é necessário, caso não esteja instalado informa com uma notificação**

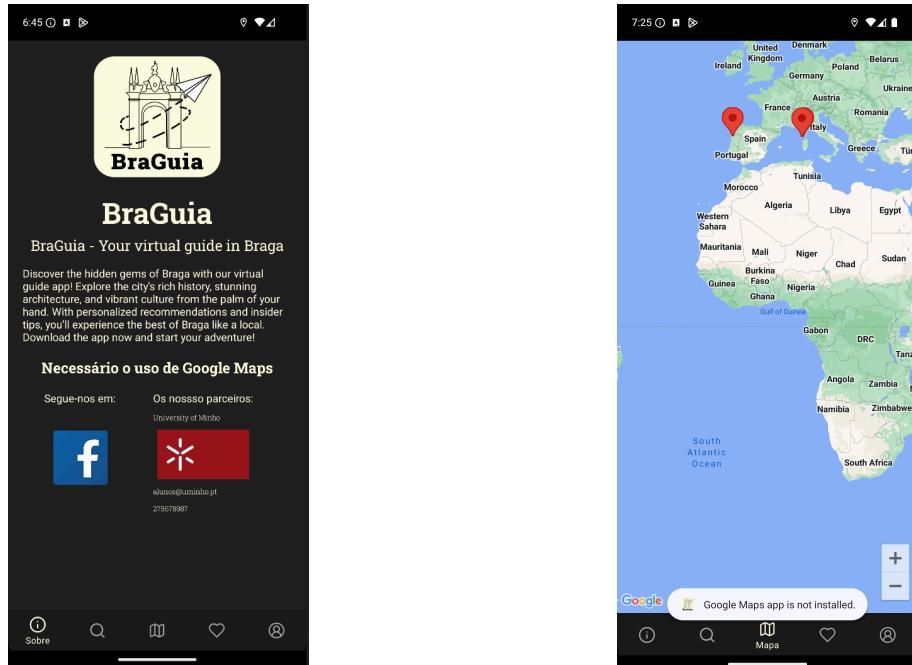


Figura 5: Requisito do Google Maps

- Para utilizadores premium (e apenas para estes) a aplicação deve possibilitar a capacidade de navegação, de consulta e descarregamento de mídia;

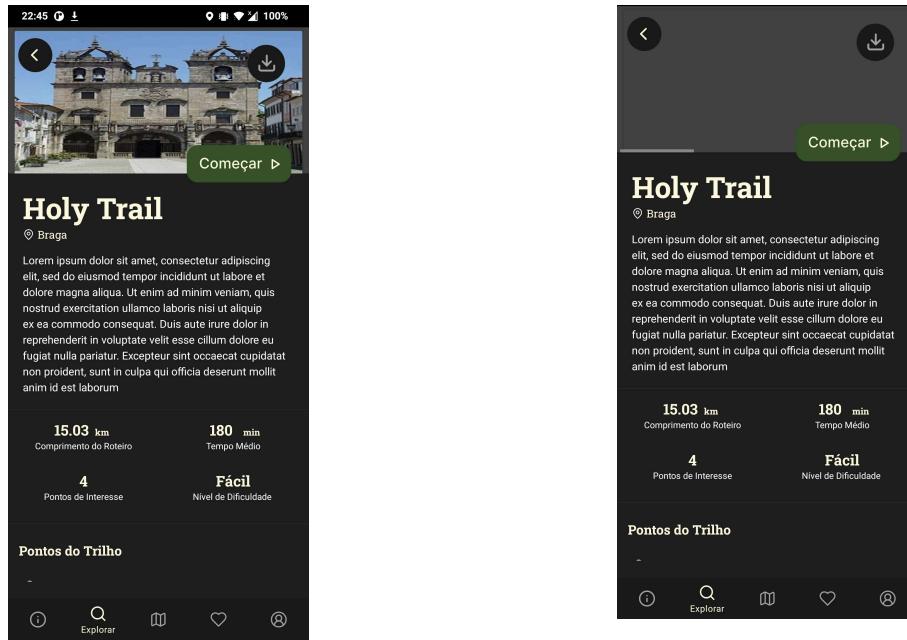


Figura 6: Premium user vs Standard User

- A navegação proporcionada pelo Google Maps deve poder ser feita de forma visual e com auxílio de voz, de modo a que possa ser utilizada por condutores; **O roteiro iniciado redireciona o utilizador para o Google Maps com as funcionalidades requisitadas.**

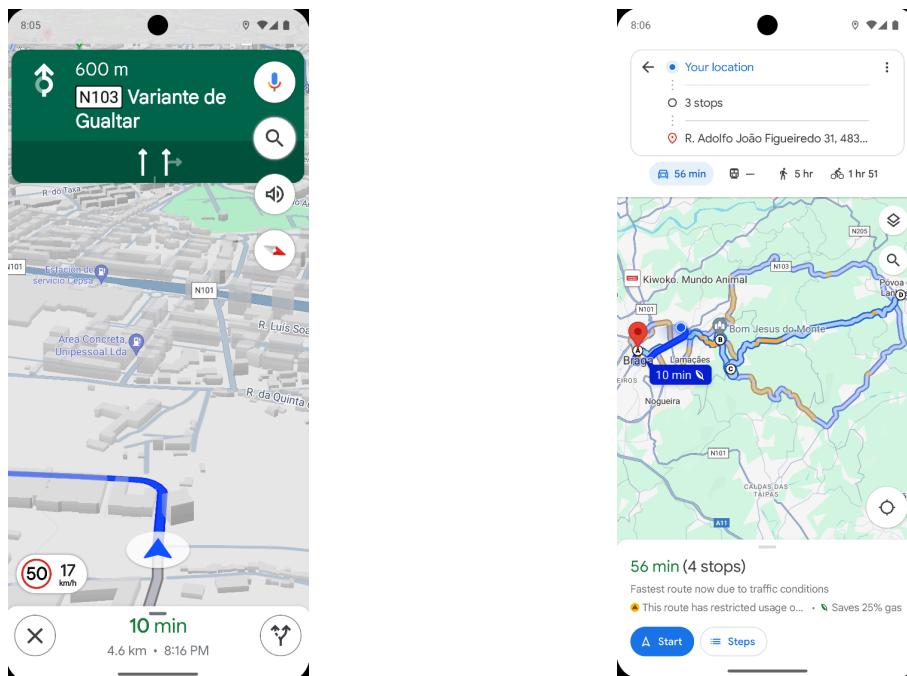


Figura 7: Roteiro no Google Maps

- A aplicação deve possuir uma página de informações acerca do utilizador atualmente autenticado;

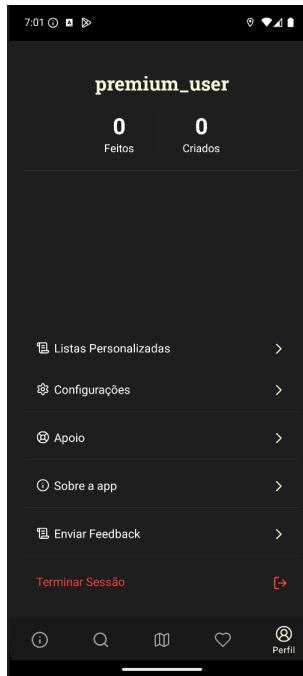


Figura 8: Página do Perfil

- A aplicação deve mostrar, numa única página, informação acerca de um determinado roteiro: galeria de imagens, descrição, mapa do itinerário com pontos de interesse e informações sobre a mídia disponível para os seus pontos;

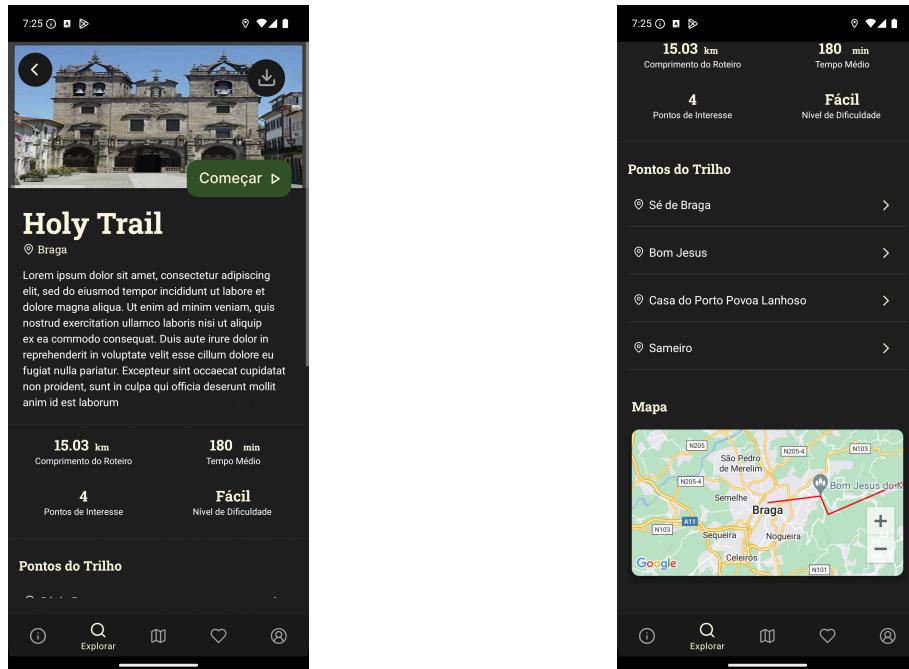


Figura 9: Página de Roteiro

- A aplicação deve possuir a capacidade de iniciar um roteiro; **Pressionar o botão “Começar” para iniciar um roteiro. Redireciona o utilizador para o Google Maps, com o roteiro pre-selecionado .**



Figura 10: Página de Roteiro

- A aplicação deve possuir a capacidade de emitir uma notificação quando o utilizador passa perto de um ponto de interesse;

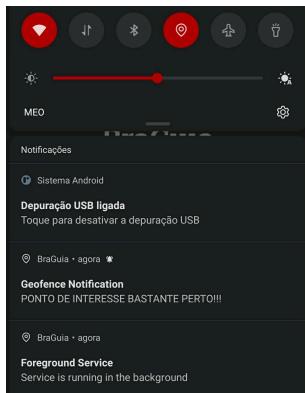


Figura 11: Notificação de um ponto de interesse

- A aplicação deve possuir uma página que mostre toda a informação disponível relativa a um ponto de interesse: localização, galeria, mídia, descrição, propriedades, etc;



Figura 12: Página de Ponto de Interesse

- A aplicação deve ter a capacidade de apresentar e produzir 3 tipos de mídia: voz, imagem e vídeo;
- A media é toda apresentada no topo da página. É possível dar slide e encontrar os diferentes tipo. O audio é iniciado quando se dá play do card respetivo.**



Figura 13: Página de Roteiro

- A aplicação deve possuir um menu com definições que o utilizador pode manipular;

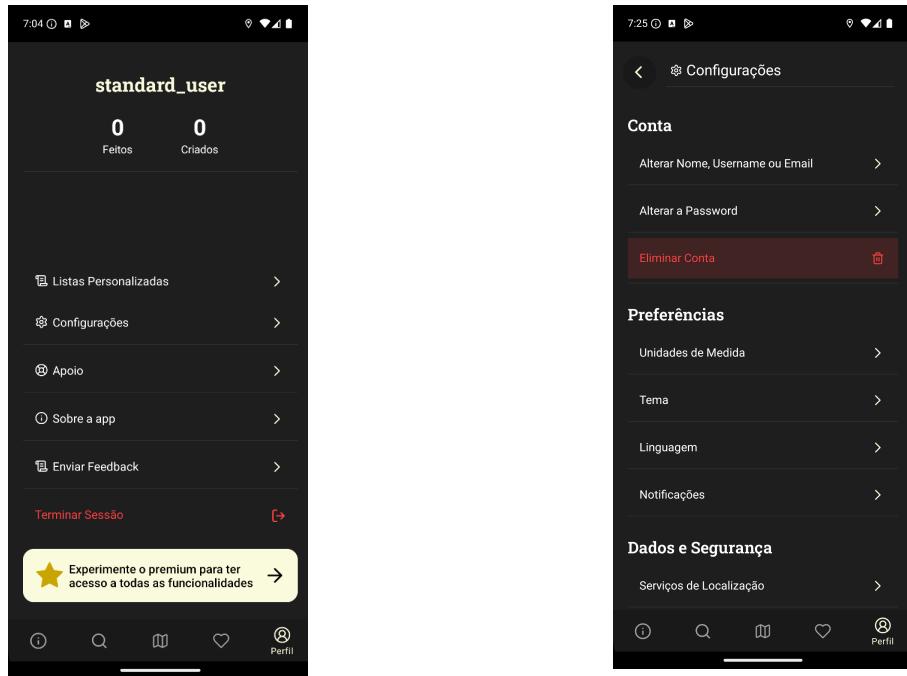


Figura 14: Página de Configurações

- A aplicação deve possuir a capacidade de ligar, desligar e configurar os serviços de localização;

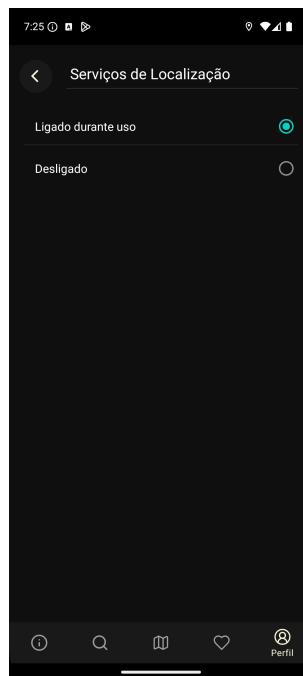


Figura 15: Página de Configurações - Serviços de Localização

- A aplicação deve possuir a capacidade de descargar mídia do backend e aloja-la localmente, de modo a poder ser usada em contextos de conectividade reduzida;

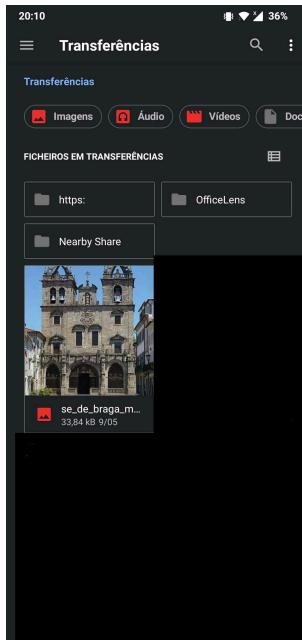


Figura 16: Confirmação do Download de media

```

IMAGEFILE      com.example.braguia2      D /storage/emulated/0/Download/se_de_braga_m0C5XV9.jpg
IMAGEFILE      com.example.braguia2      D true
CONECTIVIDADE REDUZIDA com.example.braguia2 D SAQUEI A IMAGEM LOCALMENTE
URI           com.example.braguia2      D file:///storage/emulated/0/Download/se_de_braga_m0C5XV9.jpg
  
```

Figura 17: Carregar imagem localmente

- A aplicação deve possuir a capacidade de efetuar chamadas para contactos de emergência da aplicação através de um elemento gráfico facilmente acessível na aplicação. **Acessível através da página de Explorar e do Perfil.**

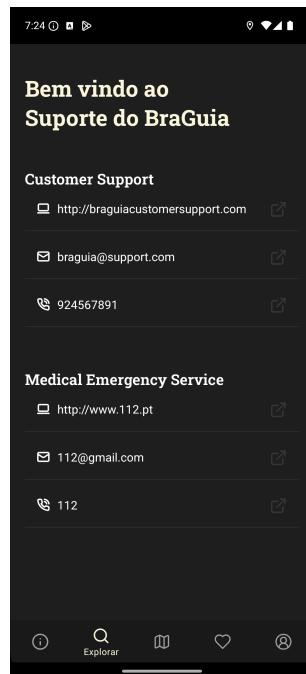


Figura 18: Página de Suporte

Discussão de resultados

Trabalho realizado

Para começar o projeto o grupo decidiu começar por usar *Kotlin*, uma linguagem orientada a objetos mais moderna e concisa do que java. No entanto, dado que o grupo demonstrou bastantes dificuldades a começar o projeto desta forma foi tomada a decisão de reverter o trabalho para *Java* utilizando o código fornecido pelo professor.

Apesar de não termos conseguido cumprir todos os requisitos funcionais descritos no enunciado, o grupo encontra-se satisfeito com o trabalho desenvolvido.

Em relação aos testes, dado que estivemos maioritariamente focados a tentar desenvolver ao máximo a aplicação e não tínhamos grandes bases no que toca a software testing, acabamos por apenas conseguir desenvolver testes white-box para classe no package model, ficando por testar a persistencia dos dados na base de dados, isto é, testar os DAOs.

Limitações

Uma limitação que o grupo encontrou, nomeadamente a construir o histórico do utilizador é que, apesar do histórico ser guardado localmente, ao abrirmos o *fragment* do histórico, é criado um novo *UserViewModel*, o que faz com que as informações dentro do *user* não tenham efeito.

Uma outra limitação é quando vamos realizar o login, de vez em quando interromper a aplicação, e ao reabrir a aplicação o login é realizado com sucesso. Quanto a este *bug*, o grupo mesmo após dar *debug*, não sabe o porquê disto acontecer.

Funcionalidades extra

Apesar de estarem parcialmente implementadas, o grupo tinha planeado desenvolver funcionalidades como:

1. O utilizador poder adicionar/tornar um trilho ou ponto de interesse favorito
2. Modo claro ou escuro
3. O utilizador poder criar um trilho personalizado ou adicionar um ponto de interesse ao trilho que está a realizar se estiver na proximidade do utilizador
4. Filtrar trilhos ou pontos de interesse consoante a sua categoria, por exemplo, trilhos dedicados à gastronomia, arte, entre outros.

Acreditamos que estas funcionalidades acima trariam a aplicação para uma aproximação de uma aplicação do mundo real e efetivamente usável.

Gestão de projeto

Gestão e Distribuição de trabalho

A distribuição do trabalho deu-se da seguinte forma:

- **Daniel Du** (pg53751)
 - ▶ UI implementation
 - ▶ App
 - ▶ Notificações
 - ▶ Location Service
- **José Pedro Fonte** (a91775)
 - ▶ Users
 - ▶ UI design and implementation
- **Ricardo Lucena** (pg54187)
 - ▶ Trilhos

- ▶ Notificações,
- ▶ Location Service

Eventuais metodologias de controlo de versão utilizadas

O grupo optou por utilizar as ferramentas GIT e Github para o controlo de versões e gestão do projeto, aproveitando a funcionalidade “Github Projects” nesta última. Inicialmente, o trabalho foi dividido e foram criadas branches individuais para o desenvolvimento das funcionalidades. No entanto, após breve consideração, a abordagem foi modificada. Dada a equipe reduzida e os possíveis obstáculos decorrentes do desenvolvimento em branches, devido às diferentes dependências, decidiu-se simplificar a metodologia e utilizar apenas uma branch principal.

O progresso do projeto pode ser consultado no [repositório do Github](#).

Reflexão sobre Performance individual

- **Daniel Du** (pg53751)

Penso que tive uma participação ativa e uma boa contribuição no projeto. Contudo, não contente de todo pois acredito que podia ter feito ainda mais.

- **José Pedro Fonte** (a91775)

Da minha parte penso que não estou totalmente satisfeito com a minha contribuição, porque apesar de ter contribuído positivamente para o trabalho acho que poderia ter ajudar mais em funcionalidades do *backend* como serviços e notificações. A minha maior contribuição foi na componente estética *UI* e *UX* tanto no design como na sua implementação, e também fiquei encarregue da funcionalidade dos *Users*, que devido a ter dado bastantes problemas, me impediu de progredir noutras funcionalidades.

- **Ricardo Lucena** (pg54187)

A meu ver contribui de forma positiva para o trabalho, apesar de não ficar 100% contente com a forma como fiz certas partes. Penso que no que toca a legibilidade e duplicação de código podia ter melhorado bastante, por exemplo. Apesar disso diria que maior parte das funcionalidades e objetivos foram alcançados, o que faz com que veja o resultado final como um sucesso.

Conclusão

Em suma, o grupo está satisfeito com a aplicação que desenvolveu.

A nível estético e de uso, está muito próximo de uma aplicação do mundo real. Sobre o backend, apesar de não estar totalmente perfeita, com mais tempo conseguiria resolver os bugs que a aplicação tem, ou até implementar a 100% certas funcionalidades em falta, como o histórico ou até a adição de trilhos favoritos ao utilizador.

Todos os elementos do grupo concordam que apesar de não termos praticamente nenhum conhecimento prévio de como usar *Android Studio*, os desafios que foram aparecendo fizeram com que tivéssemos de procurar informações na *Internet* mais afincadamente do que outros trabalhos práticos, o que fez com que o conhecimento extraído ao fazer a aplicação seja maior.

Anexos

A - Estrutura do Projeto

Directory Tree

```
./
  model
    -- App
      |-- AppAPI.java
      |-- AppDAO.java
      |-- App.java
      |-- Contacts.java
      |-- ConvertAppTable.java
      |-- Partners.java
      |-- Socials.java
    -- Favoritos.java
    -- GuideDatabase.java
    -- Trails
      |-- Conteudo.java
      |-- ConverteEdgeTable.java
      |-- ConvertePontoTable.java
      |-- ConverteTrailTable.java
      |-- Edge.java
      |-- Ponto.java
      |-- RelPin.java
      |-- TrailAPI.java
      |-- TrailDAO.java
      |-- Trail.java
    -- User
      |-- IntegerListConverter.java
      |-- UserAPI.java
      |-- UserDao.java
      |-- User.java
  repositories
    |-- AppRepository.java
    |-- TrailRepository.java
    |-- UserRepository.java
  result.html
  ui
    |-- activities
      |-- MainActivity.java
    |-- fragments
      |-- config_location.java
      |-- config_select.java
      |-- configs.java
      |-- explorar.java
      |-- favoritos_downloads.java
      |-- favoritos_fav.java
      |-- Login.java
      |-- napa.java
      |-- MediaListFragment.java
      |-- MediaListPontoFragment.java
      |-- perfil.java
      |-- PontoHistoricoListFragment.java
      |-- ponto.java
      |-- PontosListFragment.java
      |-- PontosTextoListFragment.java
      |-- PontosTraillListFragment.java
      |-- sobre.java
      |-- support.java
      |-- TrailHistoricoListFragment.java
      |-- trail.java
      |-- TraillistFragment.java
    |-- MediaRecyclerAdapter.java
    |-- PontosRecyclerAdapter.java
    |-- PontosTextoRecyclerAdapter.java
    |-- TrailsRecyclerAdapter.java
    |-- utils
      |-- GeofenceBroadcastReceiver.java
      |-- GeofencingClass.java
      |-- LocationForegroundService.java
      |-- LocationUtility.java
      |-- NotificationHelper.java
    |-- viewmodel
      |-- AppViewModel.java
      |-- TrailService.java
      |-- TrailsViewModel.java
      |-- UserViewModel.java
```